**UNDERSTANDING DARKNET’S YOLO CFG FILES:**

**Yolov4:**

YoloV4 is an improvement of YoloV3, the implementation of a new architecture completely based on Convolutional Neural Network(CNN), with the Backbone and the modifications in the Neck have improved the mAP (mean Average Precision) by 10% and the number of FPS (Frame per Second) by 12%.

**Backbone:**

It’s a deep neural network composed mainly of convolution layers.

The main objective of the backbone is to extract the essential features.

The selection of the backbone is a key step it will improve the performance of object detection.

Every image is taken as an input and features compressed down using a convolutional neural network backbone.

For every input multiple bounding boxes needs to be drawn around images along with classification, so the feature layers of the convolutional backbone need to be mixed and held up in light of one another.

The backbone network for an object detector is typically pre-trained on ImageNet classification. Pre-training means that the network's weights have already been adapted to identify relevant features in an image, though they will be tweaked in the new task of object detection.

[net]

batch=64  
subdivisions=8

**[net]:**

It indicates the neural network for which pre-trained networks are configured in order to load and ready to train.

**batch:**

It indicates the number of samples which will be processed in one batch.

**subdivisions**:

number of mini\_batches in one batch. mini\_batch = batch/subdivisions.

In the process, 64 images will be loaded at once, and then forward propagation / feed forward network will be completed 8 times. Accumulate loss to average, after 64 pictures have been forward-propagated, and then update the parameters one time later.

Subdivision is generally set to 8,16 or 32, which is a multiple of 8. The value of batch can be dynamically adjusted according to the memory usage. The sub-size can be added and subtracted at one time. Generally, the larger the batch, the better.

#Training  
width=608  
height=608  
channels=3  
momentum=0.949  
decay=0.0005  
angle=0  
saturation = 1.5  
exposure = 1.5  
hue=.1

**Width:**

network size (width), so that every image will be resized to the network size during training and detection.

**Height:**

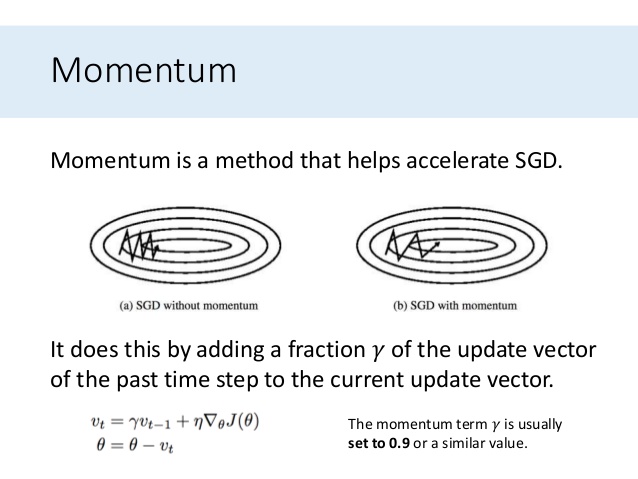
network size (height), so that every image will be resized to the network size during training and detection

**channels:**

network size (channels), so every image will be converted to this number of channels during Training and Detection. The reorganization layer reduces the size to half then creates 3 channels with adjacent pixels in different channels.

**Momentum**:

an optimization method of deeplearning which affects the stochastic gradient descent to achieve optimal speed. momentum \* previous\_gradient + (1-momentum) \* gradient\_of\_current\_batch. Makes the gradient more stable.



**Decay:**

Weight diminishing factor to avoid having large values by preventing overfitting.

It is a type of regularization method. Smaller datasets and architectures require larger values for weight decay while larger datasets and deeper architectures require smaller values. decay rate which is greater than zero.

Fixing the learning rate as too small, we would expect that a very small decay would be preferred, as a large decay would rapidly result in a learning rate that is too small for the model to learn effectively.

Large decay decay the learning rate too rapidly for a model and result in poor performance. The smaller decay values do result in better performance.

**Angle**:

Geometric distorsion/ data enhancement parameter which rotates images during training to generate more training samples by rotating the angle

**Saturation:**

Photometric distortion / data enhancement parameter which changes saturation of images during training by adjusting the saturation to generate more training samples.

Saturation augmentation adjusts how vibrant the image is.

A fully desaturated image is grayscale, partially desaturated has muted colors, and a positive saturation shifts colors more towards the primary colors.

Adjusting the saturation of an image helps the model to perform better when colors in the wild are different.

**Exposure:**

Photometric distortion / data enhancement parameter which changes exposure (brightness) during training by adjusting the exposure to generate more training samples.

**Hue:**

Photometric distortion / data enhancement parameter changes hue (color) during training to generate more training samples by adjusting the hue. Hue augmentation randomly alters the color channels of an input image, causing a model to consider alternative color schemes for objects and scenes in input images. This technique is useful to ensure a model is not memorizing a given object or scene's colors.

learning\_rate=0.0013  
burn\_in=1000  
max\_batches = 500500  
policy=steps  
steps=400000,450000  
scales=.1,.1

**learning\_rate:**

It indicates the initial learning rate for training.

The learning rate determines the speed at which the weights are updated.

The learning rate is a hyperparameter that controls how much to change the model in response to the estimated error each time the model weights are updated.

Choosing the learning rate is challenging as a value too small may result in a long training process that could get stuck, i.e. too small will make the speed too slow.

Setting too large will make the result exceed the optimal value. It may result in learning a sub-optimal set of weights too fast or an unstable training process.

The learning rate controls how quickly the model is adapted to the problem. Smaller learning rates require more training epochs given the smaller changes made to the weights each update, whereas larger learning rates result in rapid changes and require fewer training epochs.

At the beginning of training: the learning rate is preferably 0.01 ~ 0.001 often in the range between 0.0 and 1.0. The actual learning rate is related to the number of GPUs. For example, if your learning rate is set to 0.001, if you have 4 GPUs, then the real learning rate is 0.001/4.



**Burn\_in:**

The initial burn\_in will be processed for the first 1000 iterations which exponentially ramps up the learning rate from 0 to 0.001 over the first 1000 iterations.

**Max\_batches:**

Stop learning after the number of training reaches max\_batches.

**Policy**:

Learning rate adjustment strategy: constant, steps, exp, poly, step, sig, RANDOM, constant, etc.

**Steps:**

Learning rate adjustment strategy: If policy=steps, at these numbers of iterations the learning rate will be multiplied by scales factor

**Scales:**

Learning rate adjustment strategy:

current\_learning\_rate = learning\_rate \* scales[0] \* scales[1] = 0.001 \* 0.1 \* 0.1 = 0.00001

#cutmix=1  
mosaic=1

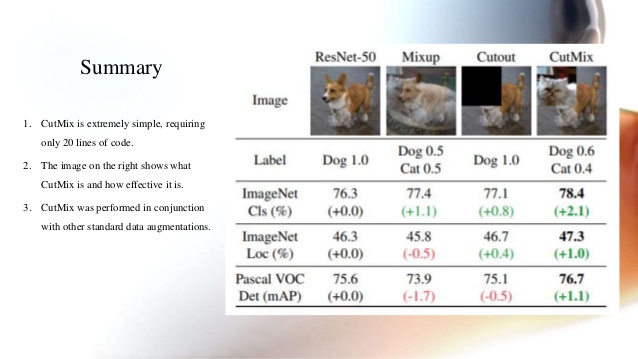
**CutMix:**

In CutMix augmentation patches are cut and pasted among training images where the ground truth labels are also mixed proportionally to the area of the patches.

CutMix improves the model robustness against input corruptions and its out-of-distribution detection performances.

Data augmentation prevents network from focusing on only few features and overfitting them.

Used for Classifier only.



**Mosiac:**

Mosaic data augmentation combines 4 training images into one in certain ratios. Mosaic is the first new data augmentation technique introduced in YOLOv4. This allows for the model to learn how to identify objects at a smaller scale than normal. It also encourages the model to localize different types of images in different portions of the frame.



[convolutional]  
batch\_normalize=1  
filters=32  
size=3  
stride=1  
pad=1  
activation=mish

**[convolutional]:**

It indicates the convolutional layer in the neural network.

A convolution is a linear operation that involves the multiplication of a set of weights with the input.

The multiplication is performed between an array of input data and a two-dimensional array of weights, called a filter or a kernel.

Using a filter smaller than the input is intentional as it allows the same filter (set of weights) to be multiplied by the input array multiple times at different points on the input. Specifically, the filter is applied systematically to each overlapping part or filter-sized patch of the input data, left to right, top to bottom.

As the filter is applied multiple times to the input array, the result is a two-dimensional array of output values that represent a filtering of the input. As such, the two-dimensional output array from this operation is called a “feature map“.’

Thus The convolution layer (CONV) uses filters that perform convolution operations as it is scanning the input “I” with respect to its dimensions. Its hyperparameters include the filter size “f” and stride “s”. The resulting output “o” is called feature map or activation map.

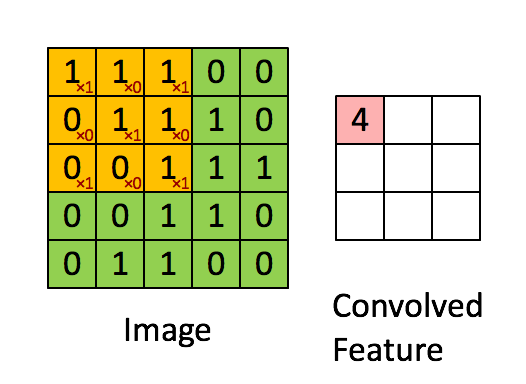


Image Dimensions = 5 (Height) x 5 (Breadth) x 1 (Number of channels, eg. RGB)

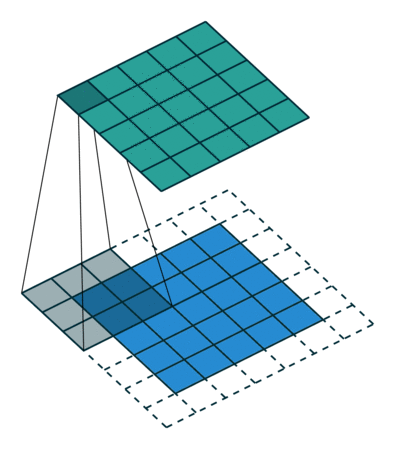
Batch normalize = Whether to perform BN processing, what is BN is not described here, 1 is yes, 0 is not. Batch normalization is a method used to make artificial neural networks faster and more stable through normalization of the input layer by re-centering and re-scaling. This has the effect of stabilizing the learning process and reducing the number of training epochs required to train deep networks.

Kernel/Filter = a 3x3x1 matrix

Stride Length = 1 means the Kernel shifts 9 times because of every time performing a matrix multiplication operation between K and the of the image over which the kernel is hovering. The filter moves to the right with a certain Stride Value till it parses the complete width. Moving on, it hops down to the beginning (left) of the image with the same Stride Value and repeats the process until the entire image is traversed.

With added layers, the architecture adapts to the extraction of High-Level features by which the convolved feature is reduced in dimensionality as compared to the input, and the other in which the dimensionality is either increased or remains the same which is achieved by padding.

Padding = amount of pixels added to an image when it is being processed by the kernel of a CNN. (Valid/Same).

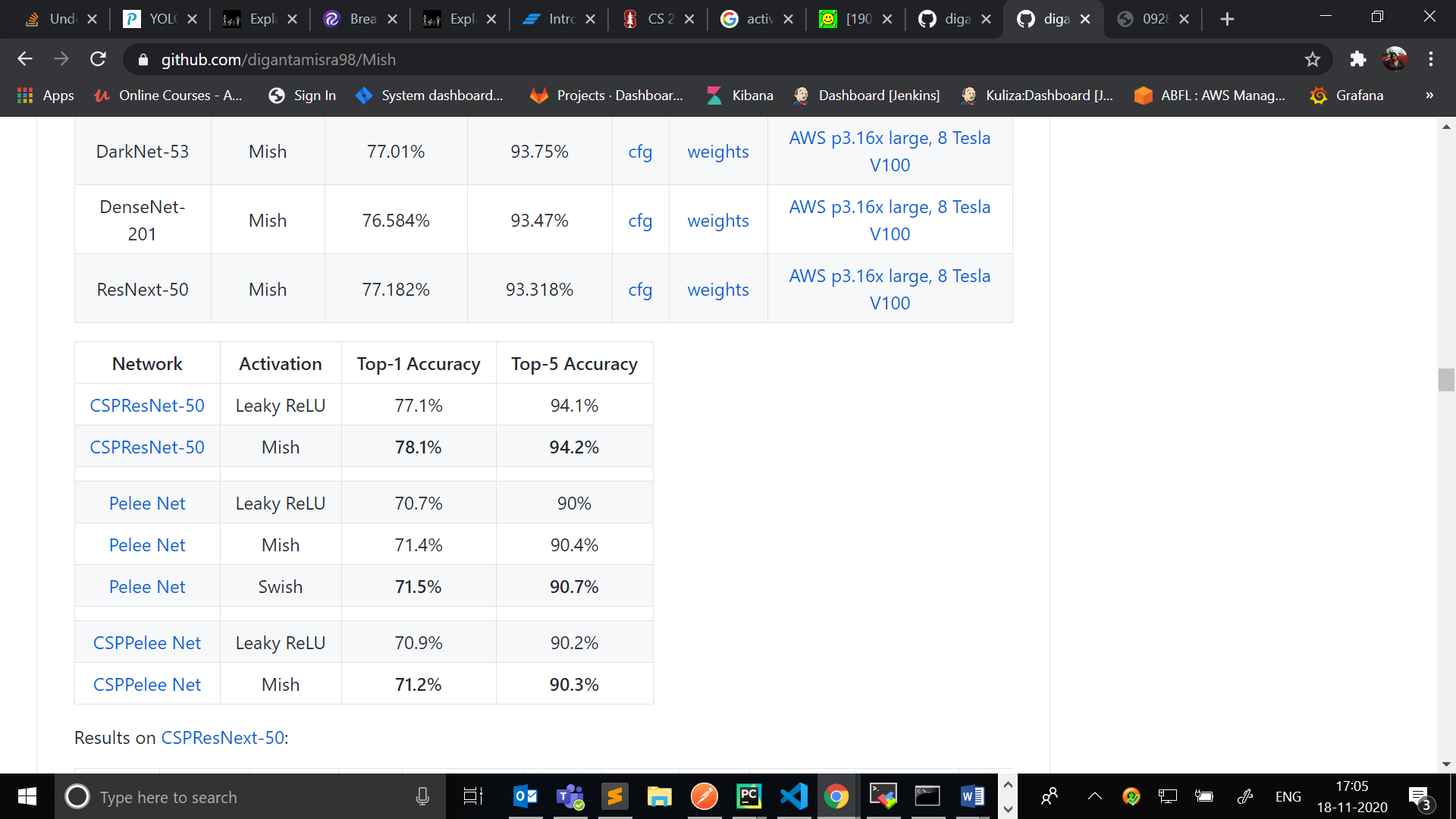


Activation(mish) = An activation function serves an integral role in the training and performance evaluation of the network.

Mish tends to work better than both ReLU and Swish along with other standard activation functions in many deep networks across challenging datasets.

F(x) = xtanh(softplus(x)) = xtanh(ln(1 + ex ))

Mish provides much better accuracy, overall lower loss, smoother and well conditioned easy-to-optimize loss landscape as compared to both Swish and ReLU. Mish has a much smoother profile than ReLU.



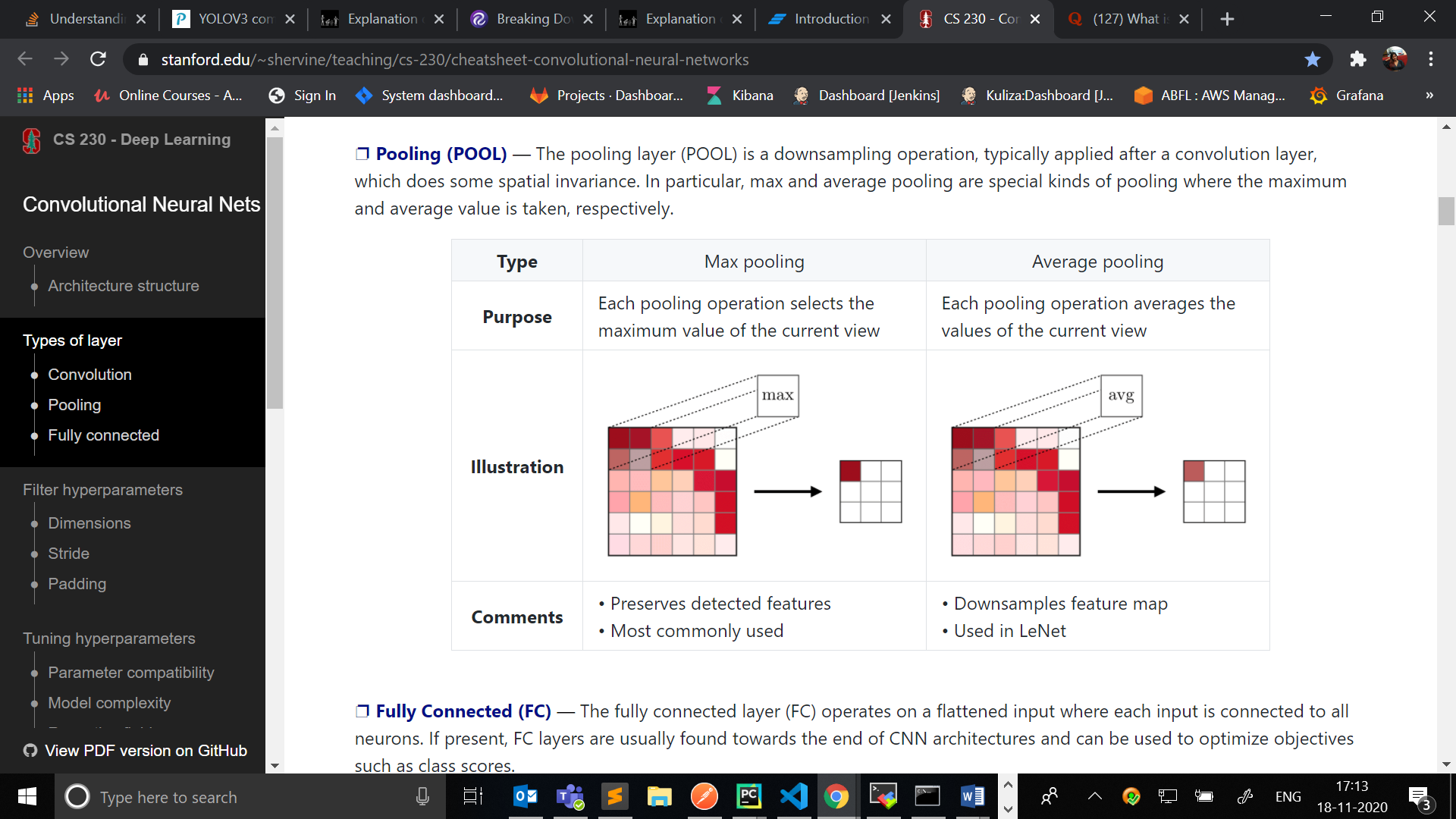
# Downsample  
  
[convolutional]  
batch\_normalize=1  
filters=64  
size=3  
stride=2  
pad=1  
activation=mish

**[Downsample]:**

The downsampling layer aka the pooling layers. It is used to reduce the height and width of the features maps as per the requirements.

A downsampling layer helps to reduce the dimensionality of the features at cost of a some loss in information. This helps save computations.

In particular, max and average pooling are special kinds of pooling where the maximum and average value is taken, respectively.



[route]  
layers = -2

**[route]:**

It is a concatenation layer, Concat for several input-layers, or Identity for one input-layer.

It is equivalent to the transfer function and does not involve convolution calculation.

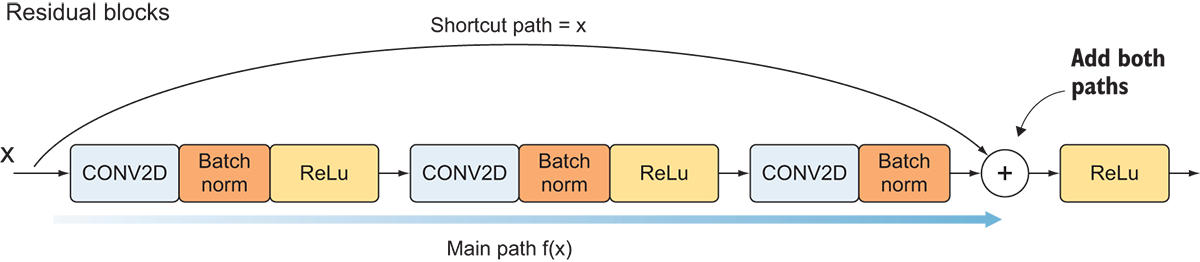
For example, in the route layer in the above figure, layer = -2 represents the feature map that leads to the convolution output of the first two layers.

It is to introduce the feature or concatenate of the previous layer and then introduce it later.

[shortcut]  
from=-3  
activation=linear

[shortcut]:

It indicates a residual connection (ResNet).



Deeper networks suffer from the degradation problem i.e. the reduction in accuracy with increasing depth of the network after reaching a maxima. And this reduction in accuracy is not due to overfitting on the training set; the training error actually starts to increase.

Short-cut connections are the connections which skip one or more layers by adding short-cut to the learned residual map.

Residual networks do take care of the degradation problem and give better performance than the plain networks. Shortcut connections have an added benefit of not adding any extra parameters or computational complexity.

-3 indicates the third layer

Linear indicates the hierarchy activation function.

[yolo]  
mask = 6,7,8  
anchors = 12, 16, 19, 36, 40, 28, 36, 75, 76, 55, 72, 146, 142, 110, 192, 243, 459, 401  
classes=80  
num=9  
jitter=.3  
ignore\_thresh = .7  
truth\_thresh = 1  
random=1  
scale\_x\_y = 1.05  
iou\_thresh=0.213  
cls\_normalizer=1.0  
iou\_normalizer=0.07  
iou\_loss=ciou  
nms\_kind=greedynms  
beta\_nms=0.6  
max\_delta=5

[yolo]:

mask = indexes of anchors which are used in this [yolo]-layer i.e3 detections (6, 7, 8).

Anchors = each grid makes 3 predictions using 3 anchor boxes, anchor\_box = predefined sized box i.e height and width of the box is defined. initial sizes if bounded\_boxes that will be adjusted during training.

Classes = number of classes in coco.names(conatins names of classes)

Num = total number of anchors. 9 anchor boxes are predefined.

jitter=.3 - randomly crops and resizes images with changing aspect ratio from x(1 - 2\*jitter) to x(1 + 2\*jitter) (data augmentation parameter is used only from the last layer)

ignore\_thresh = .7 - keeps duplicated detections if IoU(detect, truth) > ignore\_thresh, which will be fused during NMS (is used for training only)

truth\_thresh = 1 - adjusts duplicated detections if IoU(detect, truth) > truth\_thresh, which will be fused during NMS (is used for training only)

scale\_x\_y=1.05 - eliminate grid sensitivity

iou\_thresh=0.2 - use many anchors per object if IoU(Obj, Anchor) > 0.2

iou\_loss=mse - IoU-loss: mse, giou, diou, ciou

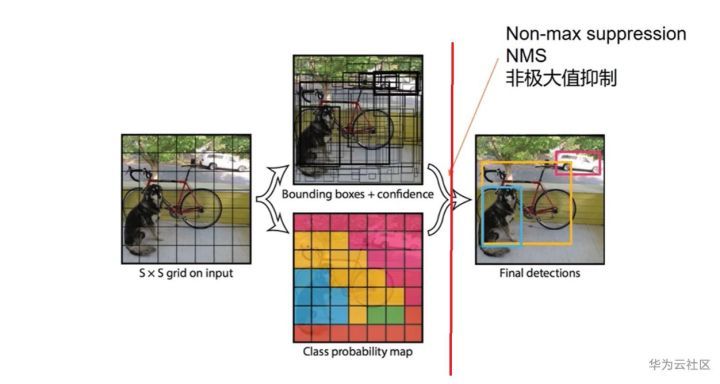
iou\_normalizer=0.07 - normalizer for delta-IoU

cls\_normalizer=1.0 - normalizer for delta-Objectness

nms\_kind= when multiple objects correspond to a single object, is solved by clustering objects by spatial closeness measured with IoU and keeping only the most confident objects among each cluster. score\_threshold: a threshold used to filter boxes by score. nms\_threshold: a threshold used in non maximum suppression.

Types : greedy NMS and optimal NMS.

max\_delta=5 - limits delta for each entry.



**Loading Network Blocks by Constructing Configuration file:**

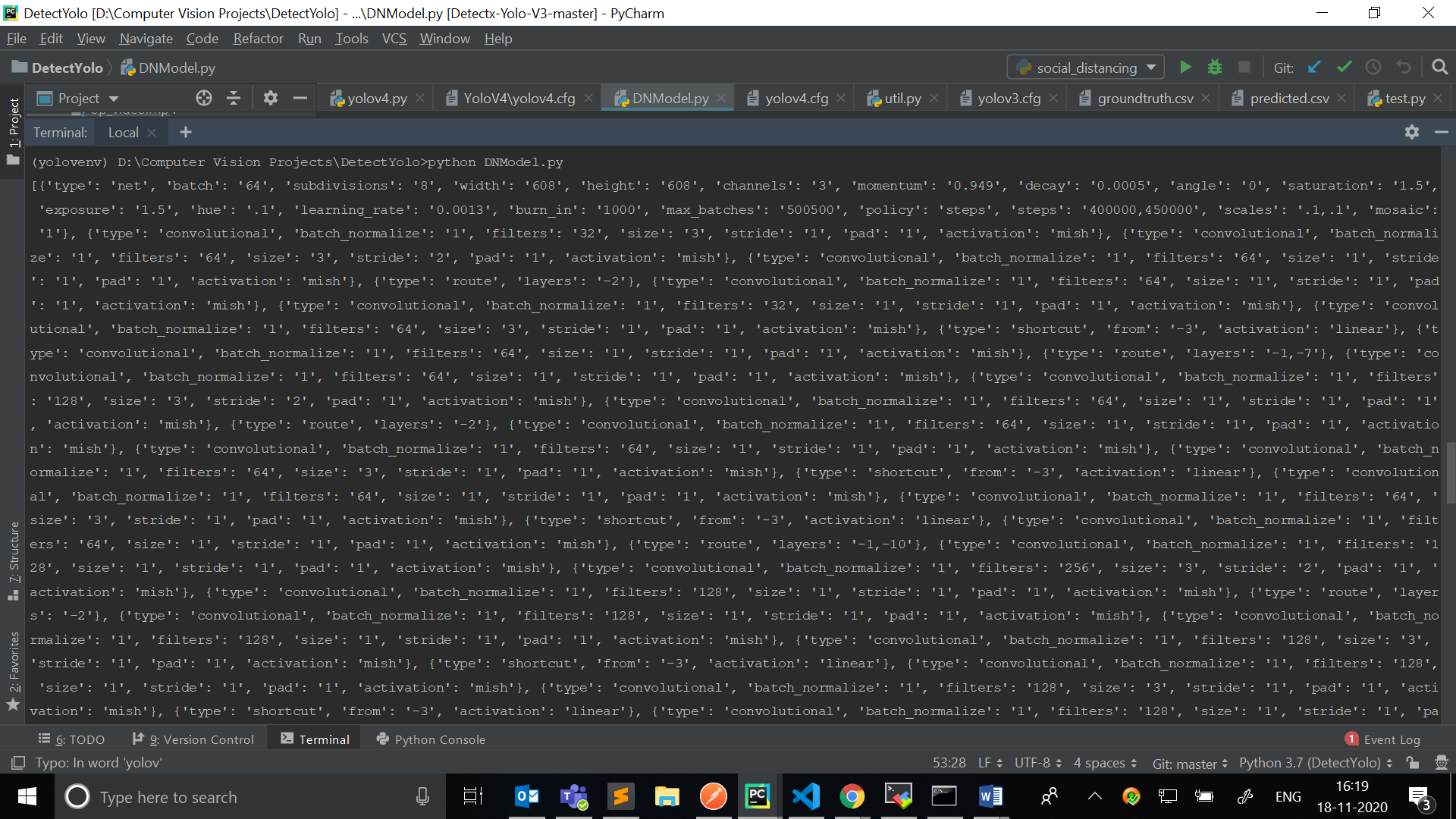
def construct\_cfg(configFile):  
 # Read and pre-process the configuration file  
  
 config = open(configFile,'r')  
 file = config.read().split('\n')  
  
 file = [line for line in file if len(line) > 0 and line[0]!= '#']  
 file = [line.lstrip().rstrip() for line in file]  
  
  
 #Separate network blocks in a list  
 networkBlocks = []   
 networkBlock = {}  
  
 for x in file:  
 if x[0] == '[':  
 if len(networkBlock) != 0:  
 networkBlocks.append(networkBlock)  
 networkBlock = {}  
 networkBlock["type"] = x[1:-1].rstrip()  
 else:  
 entity , value = x.split('=')  
 networkBlock[entity.rstrip()] = value.lstrip()  
 networkBlocks.append(networkBlock)  
  
 return networkBlocks

#Test CFG:  
construct = construct\_cfg('cfg/yolov4.cfg')  
print(construct,"/n constructed from cfg file")

# netblocks to build the network

netBlocks = construct\_cfg(cfgfile)

**Output:**



**Constructing Neural Net:**

class net(nn.Module):  
 def \_\_init\_\_(self, cfgfile):  
 super(net, self).\_\_init\_\_()  
 self.netBlocks = construct\_cfg(cfgfile)  
 self.DNInfo, self.moduleList = buildNetwork(self.netBlocks)  
 self.header = torch.IntTensor([0,0,0,0])  
 self.seen = 0  
  
 def forward(self, x, CUDA):  
 detections = []  
 modules = self.netBlocks[1:]  
 layerOutputs = {}  
  
  
 written\_output = 0

#Iterate throught each module  
 for i in range(len(modules)):  
  
 module\_type = (modules[i]["type"])

#convolution  
 if module\_type == "convolutional" or module\_type == "upsample" or module\_type == "downsampling":  
  
 x = self.moduleList[i](x)  
 layerOutputs[i] = x  
  
 #Add outouts from previous layers to this layer  
 elif module\_type == "route":  
 layers = modules[i]["layers"]  
 layers = [int(a) for a in layers]  
  
 #If layer nummber is mentioned instead of its position relative to the the current layer  
 if (layers[0]) > 0:  
 layers[0] = layers[0] - i  
  
 if len(layers) == 1:  
 x = layerOutputs[i + (layers[0])]  
  
 else:  
 #If layer nummber is mentioned instead of its position relative to the the current layer  
 if (layers[1]) > 0:  
 layers[1] = layers[1] - i  
  
 map1 = layerOutputs[i + layers[0]]  
 map2 = layerOutputs[i + layers[1]]  
  
  
 x = torch.cat((map1, map2), 1)  
 layerOutputs[i] = x  
  
 #ShortCut is essentially residue from resnets  
 elif module\_type == "shortcut":  
 from\_ = int(modules[i]["from"])  
 x = layerOutputs[i-1] + layerOutputs[i+from\_]  
 layerOutputs[i] = x  
  
  
  
 elif module\_type == 'yolo':  
  
 anchors = self.moduleList[i][0].anchors  
 #Get the input dimensions  
 inp\_dim = int (self.DNInfo["height"])  
  
 #Get the number of classes  
 num\_classes = int (modules[i]["classes"])  
  
 #Output the result  
 x = x.data  
 print("Size before transform => " ,x.size())  
  
 #Convert the output to 2D (batch x grids x bounding box attributes)  
 x = transformOutput(x, inp\_dim, anchors, num\_classes, CUDA)  
 print("Size after transform => " ,x.size())  
  
 #If no detections were made  
 if type(x) == int:  
 continue  
  
 if not written\_output:  
 detections = x  
 written\_output = 1  
  
 else:  
 detections = torch.cat((detections, x), 1)  
 layerOutputs[i] = layerOutputs[i-1]  
  
  
 try:  
 return detections  
 except:  
 return 0

**References:**

For CNN,

<https://www.analyticssteps.com/blogs/convolutional-neural-network-cnn-graphical-visualization-code-explanation>

<https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-convolutional-neural-networks>

<https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53>

For Yolov4 Introduction,

<https://www.analyticssteps.com/blogs/introduction-yolov4>

<https://becominghuman.ai/explaining-yolov4-a-one-stage-detector-cdac0826cbd7#:~:text=YoloV4%20is%20an%20important%20improvement,network%20on%20a%20single%20GPU>.

For learning rate,

<https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/>

For mish activation function,

<https://github.com/digantamisra98/Mish>

For ResNet,

<https://towardsdatascience.com/residual-blocks-building-blocks-of-resnet-fd90ca15d6ec>

For NMS,

<https://medium.com/@yusuken/object-detction-1-nms-ed00d16fdcf9>